Received 11 November 2019; revised 16 May 2020; accepted 16 July 2020.
Date of publication 23 July 2020; date of current version 4 March 2022.

Digital Object Identifier 10.1109/TETC.2020.3011663

The Self-Aware Information Processing Factory Paradigm for Mixed-Critical Multiprocessing

EBERLE A. RAMBO, (Member, IEEE), BRYAN DONYANAVARD, MINJUN SEO, FLORIAN MAURER, THAWRA KADEED, CAIO B. DE MELO, BISWADIP MAITY, ANMOL SURHONNE, ANDREAS HERKERSDORF, FADI KURDAHI, (Fellow, IEEE), NIKIL DUTT, (Fellow, IEEE), AND ROLF ERNST, (Fellow, IEEE)

Eberle A. Rambo, Thawra Kadeed, and Rolf Ernst are with the Institute of Computer and Network Engineering, Technische Universität Braunschweig, 38106 Braunschweig, Germany
Bryan Donyanavard, Minjun Seo, Caio B. de Melo, Biswadip Maity, Nikil Dutt, and Fadi Kurdahi are with the Center for Embedded and Cyber-physical Systems (CECS), University of California, Irvine CA 92697, USA
Florian Maurer, Anmol Surhonne, and Andreas Herkersdorf are with the Chair for Integrated Systems, Technische Universität München, 80333 München, Germany
CORRESPONDING AUTHOR: E. A. RAMBO (rambo@ida.ing.tu-bs.de)

ABSTRACT In order to provide performance increases despite the end of Moore's law and Dennard scaling, architectures aggressively exploit data- and thread-level parallelism using billions of transistors on a single chip, enabled by extreme geometry miniaturization. A resulting challenge is the control, optimization, and reliable operation of such complex multiprocessing architectures. Modern and future systems will be required to operate under multi-dimensional variability: from varying workload, quality-of-service (QoS) goals, and non-functional requirements to varying environmental and operating conditions. A trend has recently emerged to abstract such complex multiprocessing architectures as self-aware factories whose resources are monitored, configured and their use is planned during runtime. In this article, we present the Information Processing Factory (IPF) paradigm for mixed-criticality. We introduce its 5-layer hierarchical organization and a system configuration framework that ensures that the strict requirements of the safety-critical functions are always met while dynamically managing and optimizing the mixed-critical system at runtime. We illustrate the application of IPF in heterogeneous domains with two representative use-cases (healthcare and automotive), investigate the use of IPF to achieve long-term dependability, and highlight the open challenges. Experimental results report the reliability levels achievable with the proposed paradigm.

INDEX TERMS Self awareness, mixed criticality, real-time systems, high-dependability, reliability

I. INTRODUCTION

Modern computers exploit parallelism throughout the system stack to increase performance and efficiency [1]. The result is computer systems with complex multiprocessing architectures whose control, optimization, and reliable operation have become significant challenges. These architectures are implemented with billions of transistors on a single die, or even on multiple chiplets integrated at the package level. Throughout its lifetime, besides the threat of the usual transient, intermittent, permanent random hardware faults [2], [3], [4] and physical limitations such as dark silicon [5], [6], future systems will face multi-dimensional variability: from varying workload, QoS goals, and non-functional requirements and constraints to varying environmental and operating conditions. Thus, not only must future systems manage and optimize the resource usage at runtime, they must also cope with different types of faults, doing so while providing uninterrupted service.

The IPF project has recently introduced the abstraction of such complex architectures as self-aware information processing factories, the IPF paradigm [7], [8], [9]. These factories consist of a set of highly configurable resources, such as CPUs and interconnects, whose use is monitored, planned, and configured during runtime. Continuing with the analogy, managing a factory requires multiple considerations, such as efficiency, availability, reliability, integrity, and timing. IPF conquers the complexity of managing such systems by hierarchically decomposing the challenges. These are addressed by different mechanisms that co-exist in the factory.

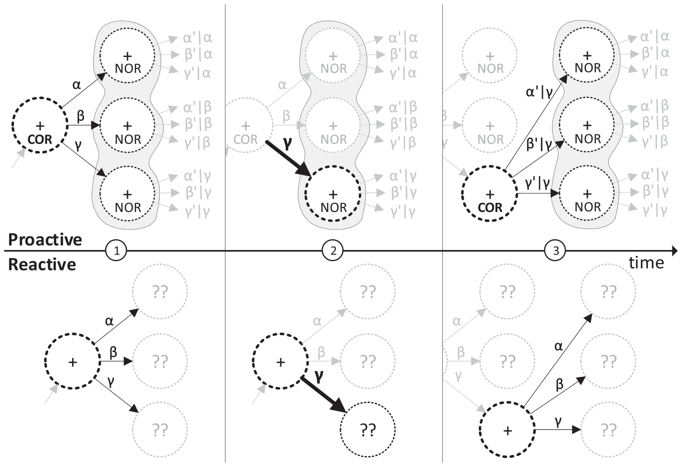

FIGURE 1. **IPF's proactive approach in time versus a reactive one.**

Future *mixed-critical real-time embedded systems* [10] will employ multiprocessing platforms to increase computational power, efficiency, and dependability. Considering the definition in [10], a mixed-critical system is a system in which application functions of different criticalities share computation and/or communication resources. Criticality can include all forms of dependability – e.g., availability, reliability, and integrity – but the term is mostly used in the context of *functional safety*. Functional safety is the absence of catastrophic consequences on the user and the environment. In most cases, safety-critical functions are also subject to *timing requirements*, such that mixed-critical systems are in most cases real-time systems [10]. Due to their direct impact on functional safety, such systems are strictly regulated by safety standards [3], [11], [12]. The standards usually define five levels of criticality – e.g., in the automotive domain, [3] defines the automotive safety integrity levels (ASILs) A through D plus quality management (QM), where QM is not safety-critical, A is the least critical, and D is the highest criticality level. Mixed-criticality is particularly difficult because it must ensure that functional and non-functional requirements of higher criticality functions are met while co-existing with less well-behaved functions of lower criticalities – the so-called *sufficient independence* between criticalities [11].

With a proactive, hierarchical approach, IPF breaks down the complexity of managing and configuring such large systems in five layers responsible for sensing, planning, optimizing, acting, and processing. Figure 1 illustrates IPF's proactive approach over time and compares it with a conventional, reactive one. An IPF system starts in an initial configuration, or a *current operating region* (COR), and proactively plans the next valid and safe configurations, or *next operating regions* (NORs) ①. When an event (γ) occurs that requires action ②, the response can immediately follow. After the reaction ③, IPF proactively plans the next configurations (NORs). In contrast, a reactive approach would only plan a reaction *after* the occurrence of an event ②. Future system configurations are unknown until a reaction is required ①③. Being proactive enables immediate reactions by IPF, resulting in a high responsiveness of the system to internal and external events – e.g., errors, and temperature and workload variations.

The self-aware IPF paradigm is an attractive solution for long-term dependability systems. IPF can tolerate, besides transient faults, multiple permanent faults. It is impractical, if not infeasible, to plan, at design time, reactions to all possible sequences of random hardware faults that might occur in the system. IPF's self-aware, dynamic management of the system enables the system to plan at runtime and handle fault scenarios as they occur. Moreover, IPF can assess the risks and proactively take action on imminent hazards to the system – e.g., a permanent fault due to wearout, increasing availability. Such dynamicity, however, presents an additional challenge if IPF is to be applied in the mixed-critical real-time domain: ensuring the system's non-functional requirements [3], [11], [12].

This paper introduces the IPF paradigm for the mixed-critical real-time domain. The contributions of this paper are five-fold:

- IPF's five-layer hierarchical organization and system configuration framework based on operating regions and operating points that enable self-awareness, self-diagnosis, self-organization, and self-optimization in mixed-criticality.
- An invariant-based safety argument that enables the use of IPF in mixed-criticality.
- Description of the open challenges in implementing IPF's self-optimization, self-diagnosis, self-organization, and maintenance mechanisms and the proactive handling of imminent hazards.
- Two representative use-cases illustrating IPF's application in heterogeneous domains: healthcare and automotive.
- Experimental evaluation of the achievable reliability gains when using IPF for long-term dependability.

The remaining of the paper is organized as follows. The IPF paradigm for mixed-criticality is introduced in Section II, followed by its invariant-based safety argument in Section III. Section IV applies the IPF paradigm for long-term dependability. The approach is explored with two representative use cases in Section V, followed by experimental results in Section VI. Finally, a review of and comparison to relevant related work is given in Section VII before concluding the paper in Section VIII.

II. THE INFORMATION PROCESSING FACTORY

IPF is a metaphor for self-aware, self-organizing, mixed-critical systems. IPF provides an infrastructure for system introspection and reflective behavior, which is the foundation for computational self-awareness. Computational self-awareness is the ability of a computing system to recognize its own state, possible actions and the result of these actions on itself, its operational goals, and its environment, thereby empowering the system to become autonomous [13].

An IPF system is defined as a hardware and software system consisting of a set of highly configurable hardware resources that execute mixed-critical workloads and whose

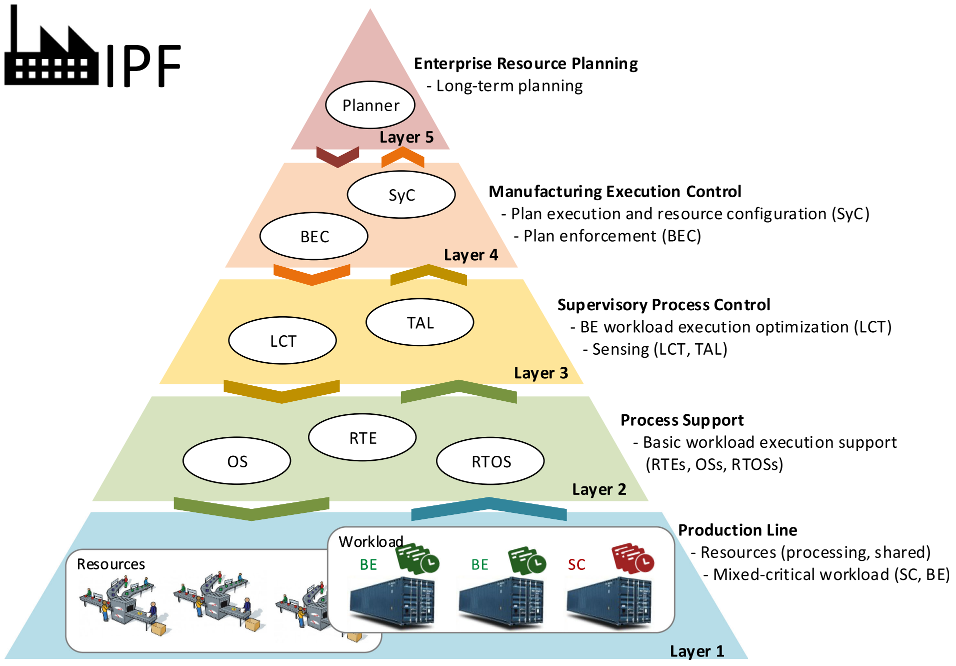
IPF
Enterprise Resource Planning
- Long-term planning
Planner
Layer 5
Manufacturing Execution Control
- Plan execution and resource configuration (SyC)
- Plan enforcement (BEC)
SyC
BEC
Layer 4
Supervisory Process Control
- BE workload execution optimization (LCT)
- Sensing (LCT, TAL)
LCT
TAL
Layer 3
Process Support
- Basic workload execution support
(RTEs, OSs, RTOSs)
OS
RTE
RTOS
Layer 2
Production Line
- Resources (processing, shared)
- Mixed-critical workload (SC, BE)
Resources
Workload
BE
BE
SC
Layer 1

FIGURE 2. **IPF's five-layer organization.**

use is planned, configured, monitored, and optimized at runtime by hardware and software components. For example, consider a network-connected pacemaker device (discussed in Section V-A) that consists of a combination of safety-critical life support functions, and non-critical network communication functions. Not only must the factory manage resources and mixed-critical workload at runtime, it must do so while ensuring that the requirements of the safety-critical functions of the workload are not violated. IPF addresses that challenge with two levels of awareness: a first level with local, autonomous actions and a second level with global ones. Together, they ensure the adaptability, safety, and dependability of the mixed-critical system throughout its execution.

Resembling a factory,[1] IPF is organized in five layers, as illustrated in Figure 2. The workload execution occurs in the *production line* (layer 1), which contains the system resources and the mixed-critical workload. The workload executes within the infrastructure and execution model of the *process support* (layer 2), which provides basic execution support such as the operating systems (OSs), real-time operating systems (RTOSs), and the runtime environments (RTEs). The resources' statuses are monitored and the workload execution is optimized by the *supervisory process control* (layer 3), which acts locally and autonomously within boundaries specified by the layers above. The *manufacturing execution control* (layer 4) is responsible for enforcing safe system configurations by globally monitoring, assessing risks, and controlling the layers below, under the guidance of the top layer. The *enterprise resource planning* (layer 5) is responsible for long-term planning of IPF. It plans future proactive and reactive actions, taking into account the operating conditions of the system, assessing risks and impacts of short-term factors such as error rates, energy consumption, workload variations; and long-term factors such as aging, energy constraints, and changes in the workload, QoS goals, and non-functional constraints. The IPF infrastructure present in layers 3-5 depends on the target application and system structure, goals, and requirements. It's possible for all or none of the layers to play a role. Consider a specific deployment of our pacemaker example that does not allow for optimization of non-critical functions. In this case, an IPF deployment could simply consist of a single layer 4 mechanism to specify the explicit resource configuration of the entire application. Note that, in the absence of the top three layers, layers 1 and 2 compose the original non-self-aware system and are always present as a minimum.

[1]Terminology inspired by enterprise control systems [14].

A. PRODUCTION LINE (LAYER 1)

The lowest layer of IPF is responsible for the workload execution. Depicted in Figure 2, the production line consists of the system resources and the mixed-critical workload.

The mixed-critical workload consists of a best-effort (BE) component and a safety-critical (SC) component. The BE workload is characterized by its goals, and the SC workload is characterized by its non-functional requirements. The BE workload has application-specific QoS, such as achieved throughput, and also has system constraints such as power budget. The SC workload has requirements, such as period, worst-case execution time (WCET), deadline, maximum downtime, maximum failure in time (FIT), and data consistency. Different levels of criticality [10] as defined in safety standards [3], [11], [12] are also supported. The five usual levels of criticality – e.g., ASILs A through D plus QM, are

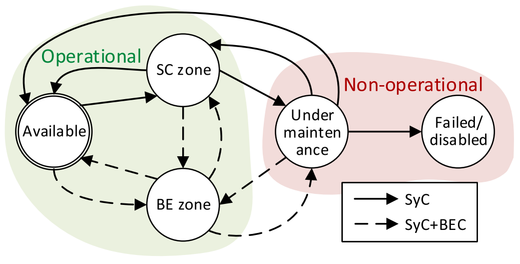

FIGURE 3. Resource states: Resource planning is done in layer 5, resource management (state transitions) is done in layer 4.

captured by the different requirements of the workload. For simplicity and without loss of generality, throughout the paper we refer to the two representative levels: BE and SC. For example, our pacemaker consists of 1) Doctor configuration, 2) AH sensing, 3) VH sensing, 4) VA activity monitor, 5) AV activity monitor, 6) APPG actuation, 7) VPPG actuation, 8) Logging, and 9) Reporting, of which 1,8, and 9 are BE workloads and the rest are SC workloads.

Processing resources are highly configurable and characterized by their properties – e.g., operating frequencies, power consumption, temperature, and error rates. They can consist of a single or multiple cores in a cluster. Processing resources can be used for executing either BE or SC workload. Therefore, IPF requires them to have a *safety-critical mode* where the execution is deterministic, predictable, and enables minimum performance guarantees for applications that require it – i.e., the SC workload. For BE workload execution, processing resources can enable non-predictable features, such as caches in the memory hierarchy and dynamic voltage and frequency scaling (DVFS). Shared resources must provide predictable and deterministic service and ensure sufficient independence[2] between different criticalities [11], thereby enabling minimum performance guarantees for the SC workload. The resources are configured by the manufacturing execution control (layer 3), which enables and disables configurable features according to the executing workload.

Processing resources in IPF have five different states and two superstates, as illustrated in Figure 3. A processing resource can either be operational or non-operational. When operational, a processing resource can be allocated to the SC workload execution or the BE workload execution. In the former case, the respective processing resource belongs to the *SC zone*, and in the latter, to the *BE zone*. When non-operational, e.g., due to errors, the processing resource can be either under maintenance or failed. If a processing resource is under maintenance, IPF attempts to find a configuration under which the resource can still be employed. When failed, the resource is no longer usable. The resource planning is done in layer 5, while resource management is done at layer 4, where the system controller (SyC) and the best-effort controller (BEC) are responsible for the different transitions. The controllers are introduced later in Section II-E. Shared resources are managed similarly and serve both SC and BE zones.

The following assumptions about the production line enable the use of IPF in the mixed-critical domain:

Assumption 1. The shared and non-shared resources in IPF present a mode in which they operate with deterministic and predictable behavior.

Assumption 2. The shared resources in IPF present mechanisms to achieve sufficient independence[2] between criticalities [11].

B. PROCESS SUPPORT (LAYER 2)

The second layer of IPF is responsible for basic workload execution support and the execution model. The process support comprises elements such as OSs, RTOSs, and RTEs, as illustrated in Figure 2. Notice that the lowest two layers alone comprise a regular mixed-critical system without any self-* properties.

For modularity and to enable predictable dynamicity at runtime, the execution model of IPF is based on containers. A container encapsulates a (sub)set of either a SC or BE workload, and is referred to as an *SC container* or a *BE container*, respectively. A container also includes an RTE with OS or RTOS, and a software stack. Each container is mapped to a processing resource, which is associated with a single container – i.e., there exists a strict one-to-one mapping between containers and processing resources. The mapping of workload to containers and the mapping of containers to resources are specified in operating regions (ORs), to be introduced in Section II-C. At runtime, the workload can be redistributed among different containers, and containers can be moved between resources, e.g., with workload balancing and migration techniques. That can be carried out by entities in layers 4 and 5 by means of transitions between ORs.

The following property of the process support enables the use of IPF in the mixed-critical domain:

Property 1. A container contains either BE or SC workload and includes the appropriate runtime environment.

C. OPERATING REGIONS AND OPERATING POINTS

Before advancing to the next layers, we define the concepts of ORs and operating points (OPs). The vision for OPs is introduced in [8]. Here, we apply the concept in a framework for configuring and managing the system. The framework consists of ORs and OPs, illustrated in Figure 4. An OR is a set of possible system configurations (the OPs) with room for optimization changes. Optimization changes are carried out by changing the OPs inside the OR. More significant changes, when required, are carried out by changing the OR.

An OR represents a configuration of the system where the mixed-critical workload (including goals and requirements), its mapping to BE and SC containers, the mapping of containers to resources, and the configuration of the shared resources are fixed. The concept is illustrated in Figure 4(a). In an OR, the configuration of the containers and the

[2]Also known as freedom from interference [3].

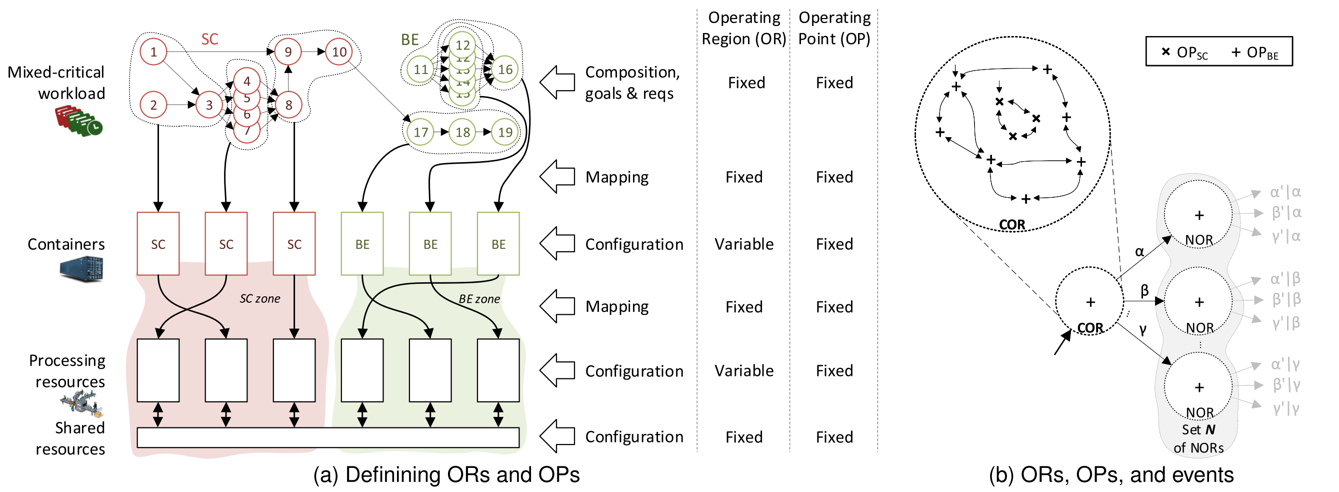

FIGURE 4. The operating region framework: current operating region (COR) and the set of next operating region (NOR), triggered by different events (α, β, and γ). An operating region can have multiple valid operating points (OPs).

associated processing resources (in the BE zone) can be varied. What can be varied depends on the specific instance of IPF and on the underlying hardware. How much it can be varied, i.e., the configuration range, is determined at runtime in layer 5, introduced in Section II-F. Both what can be varied and how much it can be varied are specified in the OR. The intuition behind ORs is that they represent valid system configurations where the system is predictable and safe for executing the SC workload, while still providing safely bounded flexibility for local optimizations of the execution of BE workload.

An OP is a specific configuration within an OR. It can also be decomposed in two components, and be defined as a pair: $\mathrm{OP} = (\mathrm{OP}_{SC}, \mathrm{OP}_{BE})$. The OP_{BE} represents a specific configuration of the BE zone, and OP_{SC} represents a specific configuration of the SC zone. The concept is illustrated in the top left quadrant of Figure 4(b), where there are multiple OPs within an OR. The intuition behind OPs is that they represent a specific, valid configuration at any given point in time within an OR.

As illustrated in Figure 4(b), the system starts at an initial, valid OR, named current operating region (COR). A number of autonomous actions can be performed locally by the system, which move the OP around inside the COR. Whenever an event occurs in IPF that requires significant changes to the configuration, IPF handles it by transitioning to a new OR, named next operating region (NOR). A suitable NOR is chosen from a set N of NORs and IPF then reconfigures the system according to the selected NOR, which becomes the COR. Then, the set N of NORs becomes empty, and new, valid NORs must be created and added to N.

Events that trigger OR transitions can have various causes, such as:

- anticipated violation of the requirements of the safety-critical workload, i.e., a hazard;
- pursuit of long-term optimization goals;
- changes in the workload, its goals and requirements;
- changes in the environmental conditions;
- changes in the operating conditions.

Events are generated by the entities in layers 3 and 4 and will be introduced in the respective layers.

The following properties of the configuration framework based on ORs and OPs enable the use of IPF in the mixed-critical domain:

Property 2. The mapping of workload to containers, the mapping of containers to resources, and the configuration of shared resources are fixed in an OR.

Property 3. The configuration ranges for the configuration of the containers and processing resources are specified in an OR.

D. SUPERVISORY PROCESS CONTROL (LAYER 3)

The supervisory process control is responsible for monitoring and autonomously optimizing the workload execution. It is also responsible for gathering useful information about the production line that supports the upper layers' long-term planning and execution. Supervisory process control components carry out actions that directly modify the configuration of the system by changing its OP. As illustrated in Figure 2, the layer comprises the IPF infrastructure components Trace Abstraction Layer (TAL) and Learning Classifier Table (LCT).

TAL is responsible for monitoring the system for errors, and it is a source of events that trigger OR transitions. TAL [15] performs runtime verification based on processor tracing. It checks the execution of the workload against contracts (system requirements) described as Timed Automata (TA) models. The contracts are loaded into TAL, which continuously monitors the system at runtime. In the pacemaker, the TAL can monitor hardware performance degradation due to aging, and propagate an increased risk of permanent failure to the IPF's layer 4 entity. TALs operate in both BE and SC containers.

The LCT is responsible for optimizing the execution of the BE workload towards achieving goals. LCTs [16] are rule-based reinforcement learning engines that explore and optimize configurations within the COR. They operate only within BE containers. LCTs collect periodic sensor data to update the fitness of rules and determine the action for the next period. First, based on the effect of the previous action toward achieving an objective, the LCT updates the rule fitness for the previous period using a version of Q-learning [17]. Second, based on the current state and rule fitnesses, the LCT applies an action to configure the system for the upcoming period by changing the OP within the COR (cf. Section II-C). For example, in the pacemaker, if it is possible to improve performance of a BE task by scheduling it more frequently without affecting SC requirements, the LCT should explore and apply this optimization for BE workloads. LCTs and TALs are configured and maintained by layer 3.

The following properties of the supervisory process control enable the use of IPF in the mixed-critical domain:

Property 4. The workload execution optimizations are bounded by the COR and concern only BE containers mapped to resources in the BE zone.

Assumption 3. Events that can affect guarantees given to the safety-critical functions are detected and reported before the guarantees can be violated.

E. MANUFACTURING EXECUTION CONTROL (LAYER 4)

In the factory analogy, the manufacturing execution control is responsible for the global monitoring, risk assessment, and control of the system. It monitors the layers below with the support of layer 3 and controls them by means of the ORs provided by the enterprise resource planning (layer 5). Changes to the system configuration initiated by this layer are realized with transitions from a COR to a NOR (cf. Section II-C).

As illustrated in Figure 2, the layer comprises two entities: the *best-effort controller* (BEC) and the *system controller* (SyC). BECSyC SyC monitors and controls the safety-critical part of the system (the SC zone) as well as the shared resources according to the COR. BEC monitors and controls the best-effort part of the system, the BE zone, according to the COR. SyC coordinates the control of the BE zone with BEC, but for safety, SyC has ultimate control over the entire system.

The SyC is responsible for configuring the system according to the COR. It configures the resources and the entities in the lower layers of IPF, loads the SC and BE containers onto the respective resources, and configures the shared resources as specified by the COR. The BEC is responsible for configuring the resources and entities in the BE zone according to the COR, and ensures that the autonomous actions carried out in the lower layers are within the specification of the COR.

Both entities globally monitor the system, assess risks, and proactively and reactively act on changes in the system or environment. These proactive and reactive measures in the system are triggered by events (cf. Section II-C), which require significant changes in the configuration of the system. Events are generated either in layer 3 or in this layer. SyC monitors the system for changes in the environmental or operating conditions that impact the execution of the SC workload. Similarly, BEC monitors the BE zone for changes in the system or environment that impacts the execution of the BE workload. Events refer to, for example, changes in the BE workload and its goals, changes in the SC workload and its requirements, or the failure of a processing resource, which will be reactively handled in IPF. An event can also refer to a predicted change that will be proactively handled by IPF, such as the imminent failure of a processing resource. Because changes in the global system configuration impact the SC workload execution, SyC is responsible for handling all events, including events that occur in the BE zone, which are forwarded by BEC to SyC. Handling the event means transitioning from the COR to a suitable NOR associated with that event. An application example of IPF with proactive handling is introduced in Section IV.

Transitions from the COR to NORs are carried out by SyC with the collaboration of BEC. Such transitions can include changes to the configuration of a single resource or can include a complete change in the mapping of workload to containers and their mapping to resources. During a transition, in addition to the above-described configuration responsibilities, resources can be added and removed from the SC and BE zones. The management of a resource's state by the SyC and BEC controllers is illustrated in Figure 3, where solid-arrow changes to the resource state involve only SyC, and dashed-arrow transitions involve both SyC and BEC. When the SC and BE zones are resized, i.e., resources are added to or removed from a zone, a controller appropriately releases its resources before handing them over to the other one – e.g., BEC removes a resource from the BE zone before handing it over to SyC to be added to the SC zone. Note that the transition of a resource from BE zone to SC zone in Figure 3 is timing critical. Therefore, SyC is allowed to forcefully execute that transition without BEC, in case the latter takes too long to release it, in order to make the execution of the SC workload independent of the execution of the BE workload (sufficient independence). Shared resources are configured by SyC since they must comply with the highest levels of criticality [3], and they must be reconfigured before any BE containers can resume execution in order to ensure sufficient independence and prevent unexpected interference on the execution of the SC workload. Note that the time to transition between ORs varies depending on the amount of reconfiguration involved.

Consider a scenario in which the pacemaker's Logging and Report BE components cause a rise in temperature significant enough to threaten the SC workload. The layer 4 components with global system view respond by spreading the best-effort workload out across more processing elements in order to dissipate heat. Layer 4 can also proactively respond to imminent threats due to aging signaled from layer 3, e.g., by migrating the threatened workload to a resource with reduced risk. The migration mitigates the increased risk to the safety-critical functions by transitioning from the current OR to a next OR.

TABLE 1. Failure scenarios in IPF where an event concerning either BE or SC workload has no associated NOR with a reactive or proactive measure.

	Reactive	Proactive
BE workload	Limited QoS (optional)	
SC workload	Failure report	Deferred failure report

When an event occurs for which there is no associated NOR, the reaction of IPF depends on the event. The possible scenarios are summarized in Table 1. In case the event is related to the execution of the SC workload, the system must signal its failure before the non-functional properties of the workload are violated. The failure is either reported immediately if the event concerns a reactive measure, or it can be deferred to when the failure actually occurs if the event concerns a proactive measure. If the event is related to the execution of BE workload, the system continues operating albeit with limited QoS, which can be optionally reported. Such scenarios can be triggered either because there are actually no more valid NORs for that event, e.g., due to a number of resource failures leading to an insufficient number of resources, or because a valid NOR has not been created yet (but eventually will), e.g., due to a quick succession of events.

The following property and assumption of the manufacturing execution control enable the use of IPF in the mixed-critical domain:

Property 5. The transition between ORs is independent of the performance of the BE workload, in the worst-case.

Assumption 4. Detected events that can affect guarantees given to the safety-critical functions are acted upon before the guarantees can be violated.

F. ENTERPRISE RESOURCE PLANNING (LAYER 5)

Finally, the enterprise resource planning is responsible for the long-term planning of the system. That is, developing the future configurations of the system in the form of NORs. The planning is supported by system information supplied by layers 3 and 4, including the resources and their current operating conditions. Planning also considers the system's current and previous ORs; current and previous operating conditions; the workload, its QoS goals and non-functional constraints; short-term and long-term factors, such as error rates, energy consumption, aging, and energy constraints; and events that may occur at runtime.

The *planner* is the main entity of this layer. Its main responsibility is to create and maintain the set N of NORs. N is modified when IPF transitions to a NOR, which requires a new set of NORs, and N is therefore emptied; and when the planner creates a new, valid NOR, in which case a new NOR is added to N. The planner also defines the valid OPs in ORs – i.e., the configuration ranges in an OR within which IPF's local autonomous actions and optimization in layer 3 can operate. That is required due to possible coupling between system resources. For example, physical temperature coupling, where the high temperature of a processing resource can affect neighboring resources and increase their error rates.

An OR is a valid configuration range of the system, and therefore the planner only includes a new NOR in the set N if it meets all non-functional requirements of the SC workload. That includes checking the ORs and their OPs with, e.g., system-level performance analysis tools such as compositional performance analysis (CPA) [18]. The planner can take the QoS goals of the BE workload into consideration, but there is no guarantee that the goals will be met in a given OR.

The transitions between different ORs are triggered by different events. Independent of the event, the planner must consider the cost of transitioning between different ORs. Transitions can involve the remapping of workload to containers and the remapping of containers to resources. Remapping requires moving code and data and therefore impacts the response time of the executing workload, which can lead to system-level timing violations (deadline misses). Thus, the planner must also check for system-level timing and safety violations of the transition from the COR to the NOR before the NOR can be included in the set N.

The following properties of the enterprise resource planning enable the use of IPF in the mixed-critical domain:

Property 6. An OR is a valid configuration for the mixed-critical real-time system where all non-functional requirements of the SC workload are met.

Property 7. A configuration is valid if all requirements of the SC workload can be guaranteed – e.g., timing, integrity, and availability.

Property 8. An NOR and its transition from a COR exist if and only if it does not violate the non-functional requirements of the SC workload at runtime.

III. THE IPF INVARIANT-BASED SAFETY ARGUMENT

Applying the IPF paradigm in a mixed-criticality system requires proof of its safety, especially for the highest criticality levels. Relevant properties have been listed throughout Section II for each layer. Now we put them together in order to formalize IPF's safety guarantees. We must cover the scenarios that may cause the violation of the guarantees given to the safety-critical functions. Given the configuration framework based on ORs and OPs, two scenarios must be considered: within a COR; and the transition from a COR to an NOR.

Lemma 1. The IPF paradigm is safe for safety-critical functions within an OR as long as Assumptions 1 and 2 are met.

Proof. The proof is by direct deduction.

From Property 2 we have that the mapping of workload to containers, the mapping of containers to resources, and the configuration of the shared resources are fixed within an OR. From Properties 6 and 7 we have that an OR meets all requirements of the safety-critical functions in the system, given that the hardware is predictable (Assumptions 1 and 2). From Property 4 we have that the performance and integrity of an SC container cannot be directly impacted by autonomous optimizations within an OR since those are limited to BE containers by design. It remains to prove that there is no indirect impact of autonomous optimizations within an OR on the SC workload through shared resources. That is ensured by Assumption 2. Thus, we conclude that the IPF paradigm is safe for the execution of the SC workload within an OR, as long as Assumptions 1 and 2 are met. □

Lemma 2. The IPF paradigm is safe for safety-critical functions when transitioning between the COR and a NOR.

Proof. The proof is by direct deduction.

The system starts in an initial OR, the COR, which is safe by definition (Property 6). The system keeps a set N of NORs, which are the next configurations that the system can take. N can only be modified in two ways: upon a transition from the COR to a NOR; or whenever the planner entity finds a new suitable NOR to be associated with an event. In the former, the set is emptied ($N = \emptyset$). In the latter, either an existing NOR associated with an event α is replaced by a better one, or a new NOR associated to a different event β is included in N. From Property 8, we have that a NOR only exists in N if the NOR and its transition from the COR are safe, that is, the performance and integrity of the SC workload are guaranteed. Since the system can only transition to a NOR that exists in N, as long as N is not empty when an event α requiring a NOR occurs, the system is safe. If event α occurs that concerns the SC workload and N is empty, the system shall indicate failure. □

Theorem 1. The IPF paradigm is safe to be applied in the mixed-critical domain, as long as Assumptions 1, 2, 3, and 4 are met.

Proof. From Lemmas 1 and 2, we have that IPF is safe for the SC workload within an OR and when transitioning between the COR and a NOR, given that Assumption 1 and 2 are met. It remains to address the event triggering the transition. From Assumptions 3 and 4, we have that events triggering OR transitions must be detected, reported and handled before any violation of the performance or integrity of the SC workload can occur. Thus, we conclude that the IPF paradigm is safe to be applied in the mixed-critical domain, as long as Assumptions 1, 2, 3, and 4 are met. □

IV. USING IPF FOR LONG-TERM DEPENDABILITY

We now apply the IPF paradigm to achieve long-term dependability of many-core platforms for mixed-critical real-time systems. In addition to self-organizing, IPF is able to assess risks and proactively act upon *imminent hazards* that threaten the system. Through self-diagnosis and maintenance, an IPF system instance can detect and handle far more complex hazards than classical error handling that rely on simple error models. In the following, we introduce a tile-based architecture template and apply the IPF paradigm. Then, we describe threats to the system dependability in the form of a fault model and detail the IPF mechanisms that ensure continuous system operation through proactive hazard detection and handling.

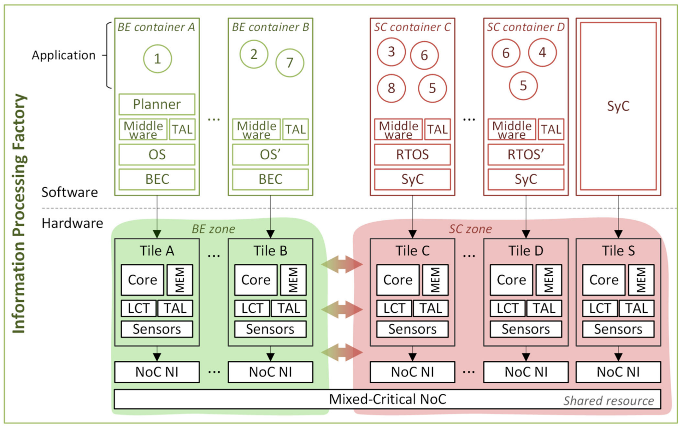

FIGURE 5. IPF on a NoC-based many-core platform.

A. THE UNDERLYING ARCHITECTURE

An architecture template is illustrated in Figure 5. The hardware architecture, depicted in the bottom half of Figure 5, consists of tiles connected by a network-on-chip (NoC). Network interfaces connect the tiles to NoC routers (abstracted as NoC in the figure). Tile contents vary: from processors and SRAMs to memory controllers, peripherals, and I/O interfaces. The components of a tile are connected to each other and the network interface by a local bus. In IPF terminology, tiles with compute units are considered processing resources; the NoC, memory controllers, and I/O interfaces are shared resources.

The processing tiles can be homogeneous or heterogeneous. For simplicity, in this paper we consider homogeneous tiles that include at least one processor core, an LCT instance, and a TAL instance. They also include sensors for monitoring power consumption and temperature. The LCT and TAL instances are hardware implementations of the entities introduced in Section II-D. LCT is a rule-based implementation of a reinforcement learner to adapt to system changes. Each LCT instance consists of sensors for determining state and evaluating the objective function, logic to evaluate the objective function and calculate reward, a rule table to store rules, and actuators to modify the system configuration. LCTs operate only within BE containers, and are disabled in SC containers. TAL is a non-intrusive runtime verification technique based on timed automata [19], which monitors and verifies properties at runtime. TAL's hardware implementation consists of a compatibility layer that provides a compatible interface for different processors, a filtering layer that has a programmable filtering mechanism to extract information needed for the verification, and a verification layer which checks system properties at runtime. The TAL is connected to the local bus and also to the processor via its trace interface, and it can operate in both BE and SC containers. The verification properties are derived at design-time [15].

The platform must be predictable and provide sufficient independence between criticalities in order to meet Assumptions 1 and 2 and to support mixed-criticality [10], [11].

These requirements are satisfied by spatial isolation and bounded interference in shared resources. Spatial isolation is achieved with the tile-based architecture together with a NoC for mixed-critical real-time systems [20]. The NoC implements wormhole switching, where variable-sized packets are composed of fixed-sized flow control units (flits); virtual-channel flow control, where flits transit through a number of virtual channels; priority-based arbitration; and deterministic source routing, where the route and virtual channel are defined in the network interface. The network interfaces control the NoC communication on a whitelist basis: tiles can only communicate with other tiles and resources if they are allowed by the SyC. Moreover, the network interface also can enforce spatial isolation by means of a memory protection unit with memory translation. The configuration of the NoC network interfaces is maintained by the SyC.

The software architecture is depicted in the top half of Figure 5. As described in Section II-B, the mixed-critical workload is partitioned into BE and SC containers that are mapped to processing tiles at runtime. Each container includes the runtime environment with operating system and the software stack for the execution of the respective workload. The container can be tailored for its workload. For example, an SC container includes an RTOS for safety and predictability, whereas different BE containers can include a different OS and software stacks for efficiency and performance.

B. IMMEDIATE ERRORS

1) HARDWARE FAULT MODEL

Critical integrated circuit design today protects hardware platforms against immediate errors caused by random hardware faults [3], also known as physical faults [21]. We call the resulting errors *immediate* because they have an immediate effect on the system state, even if they are still latent and have not yet been detected. Faults, when activated and not masked, cause errors [21]. An error has different effects on different layers of hardware and software [4]. If not appropriately detected and handled, the error propagates and can cause system failure [21] – i.e., the discontinuation of operation.

Random hardware faults can be transient, intermittent, or permanent. In hardware, transient and intermittent faults are usually abstracted as bit-flips [4]. Transient faults are caused by electromagnetic interference and energetic particles. They occur once and subsequently disappear. Intermittent faults occur from time to time and then disappear. They can be confused with transient faults, but they tend to occur in bursts at the same location, and the replacement of the affected circuit removes the intermittent fault [22]. Intermittent faults can be caused by circuit-level timing violations during runtime due to process variability, temperature variations, and aging. Permanent faults are events where the device fails permanently. They usually occur at the beginning or end of the lifetime of the device, due to manufacturing defects or aging processes, respectively. In hardware, permanent faults are usually abstracted as stuck-at or bridging faults [22]. Before the occurrence of permanent faults caused by aging processes, increasing error rates and intermittent faults are observed [22], [23].

2) DETECTION AND HANDLING OF IMMEDIATE ERRORS

Error detection and recovery in mixed- and safety-critical computation usually relies on modular redundancy approaches applied in time or space. Error detection in communication heavily relies on information redundancy. For achieving integrity and predictability, it is therefore imperative that errors be detected and contained before they propagate. Moreover, they must be handled before the system availability is compromised and the timing constraints of the SC workload are violated. That includes silent data corruption – i.e., undetected faults present in the system, which can lead to multiple error scenarios and system failure. Most practical fault-tolerant systems are dimensioned for single error scenarios, with the assumption that an error will be handled before a second one occurs.

The current state of the art includes the so-called cross-layer approaches, which combine error detection and handling techniques in different layers of the system stack for lower overhead and increased efficiency. An example is the redundant software execution with hardware-supported error detection [24], where the software can be protected with replication in space or time, for example in dual modular redundancy (DMR) or triple modular redundancy (TMR), combined with efficient error detection in hardware. The approach can be additionally coupled with other error detection mechanisms to increase coverage and resilience while ensuring integrity [25]. Another example is TAL, introduced in Section II-D and integrated in IPF as a layer-3 entity. TAL is a runtime verification technique based on timed automata [19]. It continually checks whether the workload execution satisfies (or violates) certain properties.

Like a factory, IPF also allows *self-diagnosis* at a higher level, which enables the extended local diagnosis of isolated chip resources at runtime, as commonly prescribed by safety standards [3]. Coupled with TAL, self-diagnosis can be time-triggered. Online self-diagnosis can be periodically executed on BE and SC containers within the COR. Self-diagnosis with reconfiguration, more thorough and less frequent than online testing, can be performed periodically by transitioning to a specific NOR. If the diagnosis finds a suitable configuration for the affected tile that is still suitable for an SC or BE container, the system can transition to a NOR where that tile is operational again. That is represented by the *under maintenance* resource state in Figure 3. After diagnosis, either the resource is deemed operational again or permanently non-operational (failed).

Handling immediate errors is a reactive activity, since the error has already occurred, and tolerating them requires redundancy in some form. The industrial approach for safety-critical systems still relies on costly hardware modular redundancy approaches, such as diverse DMR or TMR with lock-step

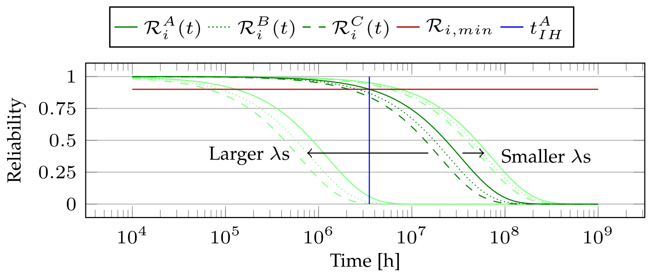

FIGURE 6. Imminent hazard as a reliability drop at time t_{IH}^A for a critical function i in a given OR A. For comparison, the reliability in ORs B and C on a same system and on systems that have aged faster (larger λs) and slower (smaller λs).

execution [26]. State-of-the-art cross-layer approaches, such as [24], resort to error handling and recovery in software, which allows for larger flexibility and less overhead.

C. IMMINENT HAZARDS – A NEW HARDWARE FAULT MODEL FOR LONG-TERM DEPENDABILITY

1) HARDWARE FAULT MODEL

The classical hardware fault model outlined in Section IV-B1 captures changes of state that have already occurred. The detection and handling, if possible, of such faults must be done before the error propagates, in order to ensure integrity, and before a failure occurs. This requirement can be hard to meet if the effects are only visible very late, as in silent data corruption, or spread quickly, as in operating systems [27]. Furthermore, physical effects, such as aging, can increase the susceptibility to random hardware faults in a way that, in combination with late error detection, the single error assumption does not hold any more.

In critical systems, we are interested in the resulting reliability – i.e., the probability of survival. The reliability in time $\mathcal{R}(t)$, also known as the survival function, gives the probability that the system survives the time interval $(0, t]$ and is still functioning at time t [28]. Using the reliability in time rather than fault probability gives us the opportunity to cover dynamics in the fault probability, such as the resulting effects of different aging rates and process variability. Even more importantly, given the required minimum reliability $\mathcal{R}_{min}$, the timing constraints of a critical function, and the hardware fault probability, we can compute the maximum time t_{max_repair} the system has to take action. The concept is illustrated in Figure 6, which shows the reliability in time $\mathcal{R}_i^A(t)$ of a critical function i in an OR A and the minimum reliability $\mathcal{R}_{i,min}$ required by that function. To abstract from the many potential physical effects affecting hardware faults and, consequently, reliability, we introduce the concept of an *imminent hazard*.

Let us start with the hazard. A hazard is a consequence of the system failing to function as expected. From ISO 26262 [3], a failure is the termination of the ability of the system or sub-system to perform a function as required. A hazard is then a potential source of harm caused by the malfunctioning behaviour of the system [3], i.e., a system failure. An imminent hazard is then defined as follows:

Definition 1. Imminent hazard: an increased risk of future errors that can lead to system failure and, therewith, a hazard.

An imminent hazard can be caused by physical causes, environmental conditions, or operating conditions. They can be caused, e.g., by the imminent failure of a resource due to an impending permanent fault. When a permanent fault is close to occur due to aging processes, increasing error rates and intermittent faults are observed [22], [23]. In this case, imminent hazards must be distinguished from latent faults and regular fault occurrences. Imminent hazards can also be caused by error rates for which the system is not dimensioned to tolerate.

Now, we can formally define an imminent hazard in terms of reliability. An imminent hazard occurs if, any time t, $\mathcal{R}_i^A(t) < \mathcal{R}_{i,min}$, where A is a given OR and i is a critical function. In other words, the probability or risk of a hazard becomes higher than acceptable. That is illustrated in Figure 6, occurring at time t_{IH}^A. In practice, a conservative hardware fault probability estimation providing a lower bound on $\mathcal{R}_i^A(t)$ will be sufficient for a conservative imminent hazard estimation and timing (proof is straightforward). This imminent hazard model will be the basis to extend detection and handling of immediate errors to a strategy for long-term dependability.

2) DETECTION AND HANDLING OF IMMINENT HAZARDS

The detection of imminent hazards differs from detection of immediate errors because there is no change to the state of system yet. Besides passive observation of untypical physical parameter values, there are proactive ways of system identification or active parameter testing, both to be executed without interruption of service and with no reduction of reliability, just like in a factory. One way to predict imminent hazards is to employ a stochastic model that reflects the reliability of the platform. An imminent hazard represents an increased risk that requires action in bounded time, which is nicely reflected by the reliability metric. As the reliability of a safety function in a given platform configuration (OR) decreases and falls below the minimum required, the imminent hazard is detected and reported as an event to the SyC.

Ideally, the next imminent hazard of a critical function in a given OR can be predicted by evaluating the model. However, no two platforms age identically, even when the design is identical. Factors, such as process variation, temperature, and switching activity resulting from the executed workload, result in different aging of different instances of a same platform, or even different aging of identical resources within a same platform instance. Thus, to accurately reflect the platform's reliability, the model must be regularly updated. The model is updated by periodically characterizing the platform resources, and updating their failure rates λ, which capture the resources' aging process, in the model. The characterization can be integrated with the online testing and the self-diagnosis with reconfiguration introduced in Section IV-B2. Figure 6 illustrates what happens to the model is updated with larger λs and smaller λs, which shift the reliability curves to the left and to the right, respectively. Although, a

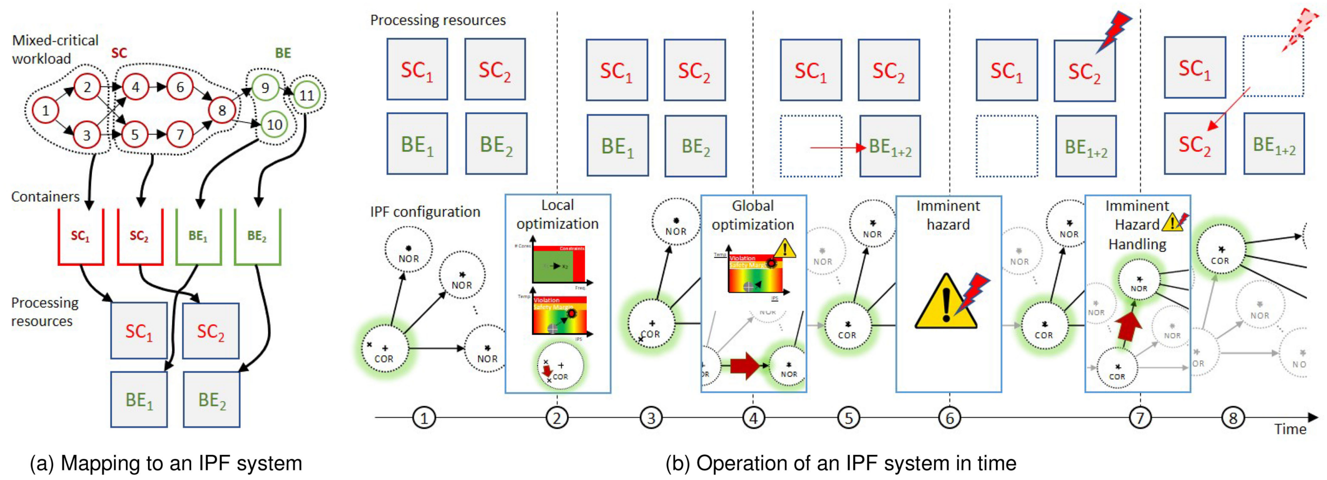

FIGURE 7. Example of a healthcare pacemaker application with a mixed-critical workload mapped to an IPF instance (a). The operation of that IPF system in time and its configuration by means of operating regions (ORs) and operating points (OPs) (b).

conservative model can be safely employed, an accurate, self-updating model can reduce costs of overprovisioning for safety and remove the dependence on correct design-time estimations.

Complementary to the stochastic reliability model-based detection, a prediction mechanism that predicts imminent hazard occurrences can be employed. With the complementary prediction mechanism, a less conservative $\mathcal{R}_{i,min}$ can be employed, such that, an imminent hazard occurs earlier than predicted (t^A_{IH}) can be safely detected. Such a mechanism can be an extension of TAL [15], which monitors parameters of the system and predicts the imminent occurrence of a hazard. An initial idea is introduced in [9].

Factory operation manages such risk-increasing effects in their devices by employing maintenance and repair to bring the failure probability back to an acceptable level, i.e., within range of the dependability requirements. Similarly, maintenance is an intrinsic part of safety standards. In contrast to factories, on-chip system repair by replacing degraded parts is not possible. Instead, reconfiguration and reorganization techniques can be adapted and combined into self-organization that proactively handles imminent hazards to functions of different levels of criticality.

The proactive self-organization handles the imminent hazard by transitioning the system to a new configuration (NOR) that increases the reliability of the critical function to the required level. To achieve that, in new configuration (e.g., OR B), the affected SC workload can be remapped to resources that are less probable to fail. Platform resources with moderately higher failure probabilities that are unfit for SC workload anymore might still be acceptable for BE workload. Special configuration for those resources can be specified in the NOR, e.g., limited temperature or voltage.

The transition between configurations can involve the migration of tasks and containers, which is timing critical due to the downtime involved. As stated in Section II-C, not only the ORs must meet the requirements of the SC workload, but also the transitions between ORs. In practice, those transitions involve a set of mechanisms and protocols between the SyC and BEC that are timing critical.

Finally, the proactive handling of imminent hazards relies on the NORs provided in advance by the planner. The specific NOR to be chosen depends on the actual hazard. Note that predicting and computing all these possibilities at design time might be unfeasible, if not very limiting, because the system can only react to known scenarios and does not support, e.g., any type of unsupervised changes to the workload. Besides imminent hazard detection, the reliability model can also be used by the planner to improve the planning of NORs since it reflects the current condition of the system resources. The idea will not be explored further here.

V. APPLYING IPF IN HETEROGENEOUS DOMAINS

This section illustrates the application of the IPF paradigm in heterogeneous domains. We use two representative use cases from the healthcare and automotive domains.

A. HEALTHCARE DOMAIN

Our first representative use case is a network-connected pacemaker.[3] The pacemaker supports physician-configurable pacing configuration, the communication of cardiac activity to a home monitoring device, and emergency physician notification for significant cardiac events. The pacemaker application consists of a mixed-critical workload of eleven tasks. Eight SC tasks are responsible for the core pacemaker functionality, which involve monitoring the heart, and actuating when required. These tasks must execute under tight and strict timing constraints and reliability. Three BE tasks implement the functionality that is not life-critical: the configuration interface with the physician and interface for analyzing the cardiac data.

[3]The example was first introduced in the special session paper [9].

The SC tasks need to execute continuously in order to guarantee that the required pace is maintained. Although the precise deadlines for actuation vary upon each patient, the software timers must be triggered every 1 ms to allow appropriate action in a timely manner for all circumstances. Additionally, given that the pacemaker execution is critical and must monitor the patient's condition indefinitely. For the timing requirements for application, Atrio-Ventricular Interval, Ventriculo-Atrial Interval, Ventricular Refractory Period, and Post Ventricular Atrial Refractory Period timers for interval calculations have default deadlines of 150 ms, 850 ms, 620 ms, and 350 ms, respectively. These interval values are later adjusted according to the arrhythmia of the patient. The life expectancy of a typical pacemaker is at least five years, which involves over 157 billion reliable timer operations [15].

Figure 7 illustrates the mapping of a pacemaker application to an IPF tile-based many-core system, as described in Section IV-A, and exemplifies the different features of IPF. IPF verifies the execution of safety-critical workload at runtime, it proactively acts upon imminent hazards, and it also optimizes both locally and globally the best-effort workload execution.

The pacemaker can be mapped to an IPF system, as illustrated in Figure 7(a). The mixed-critical workload, represented as a directed acyclic graph (DAG), consists of eight safety-critical tasks (red nodes) and three best-effort tasks (green nodes). The arrows represent data and control dependencies between tasks. In the initial configuration, the workload is partitioned into two SC and two BE containers, which are mapped to four tiles (processing resources), as indicated by the arrows. Notice that some details, such as software stack and the NoC, are abstracted away in the figure for clarity.

A possible execution sequence of that IPF system is illustrated in Figure 7(b). The system starts in an initial configuration ① within the range of the COR. Local changes in the configuration of containers and resources move the OP within the COR. At time ②, workload variations are detected in one of the resources (BE_2) by layer-3 entities, who identify opportunities for local self-optimization. The local self-optimization is then carried out in ③ by changing the configuration of the resource (executing workload of container BE_2) according to a given goal, represented by a change of the system's OP within the same COR. Later at time ④, a more significant workload variation is detected by the layer-4 global monitoring of the system. The global self-optimization and self-organization are carried out at ⑤ by moving the best-effort workload (containers BE_1 and BE_2) into a common processing resource (combined container BE_{1+2}) and powering-off one of the resources to save energy. That is represented by the transition from the COR to a NOR. Later on ⑥, an imminent hazard caused by an impending permanent fault due to aging processes is detected by IPF's layer-3 self-diagnosis. It represents an increased risk to the system. Upon detection, IPF's self-organization takes a proactive measure to mitigate the increased risk to the safety-critical functions in the workload ⑦. The proactive action is carried out by migrating the affected safety-critical workload (SC②) to a resource with reduced risk, represented by a transition from the COR to an NOR ⑧.

B. AUTOMOTIVE DOMAIN

Our second representative use case is in the automotive domain, which is currently experiencing disruptive advances in both embedded hardware and software. Until recently, such systems have been statically configured, as seen in AUTOSAR classic [29], with a configuration defined offline at design time and uploaded to the car's electronic control units (ECUs) during maintenance at a mechanical workshop. Software updates were seldom and time consuming.

Modern automotive systems are now complex distributed systems with heterogeneous ECUs [30] connected by multi-domain, heterogeneous networks [31]. Future applications to be executed on these systems have introduced new complexity levels that are not feasible with current electric and electronic (E/E) architectures. Consider advanced driver assistance systems and autonomous driving applications. They consist of several functions: localization, perception, planning, control, and system management [30]. From those, perception [32] and planning [33] are particularly complex and have stimulated the introduction of new high performance, efficient multiprocessing ECUs with hardware accelerators for, e.g., convolutional neural networks (CNNs). Tasks of these complex applications are then mapped to multiple, distributed ECUs to across the system, in order to manage the amount of data sensed, transmitted, and processed. In contrast to their predecessors, these connected systems will be much more frequently remotely updated with, e.g., security patches and algorithm improvements.

As part of an automotive system, ECUs must present certain reliability levels depending on the criticality of the function it implements. These are defined by standards such as the ISO 26262 [3], which define criticality levels – from the most critical, ASIL D, to the least critical, QM – as well as the Hazard Analysis and Risk Assessment methodology to assign ASILs to functions, considering factors such as the exposure, severity, and the controllability of the vehicle in case of function failure. The hardware failure probabilities defined by [3] as target values, and by [11] as requirements, are between 10^{-8} failures per hour or less for ASIL D and 10^{-7} for ASILs B and C. ASIL A and QM do not have target reliability values, and consider reliability as QoS.

In our use case, we focus on one multiprocessing ECU of such a system. The ECU consists of an IPF system, to which the perception tasks are mapped. For such a task, we consider a larger system with 32 processing resources that consist of clusters of cores. Similarly to the pacemaker example illustrated in Figure 7, the mixed-critical workload is mapped to

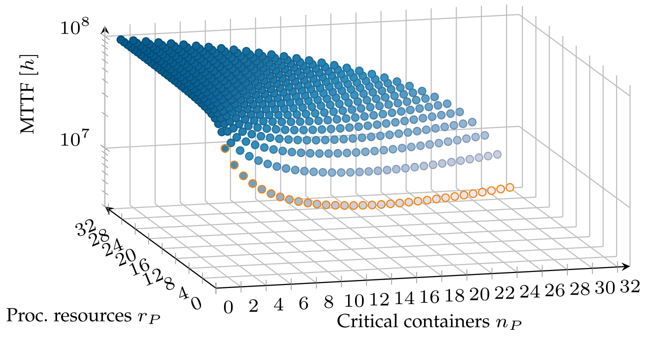

FIGURE 8. MTTF of an IPF system when varying the degree of redundancy. For reference, baseline ($r_P = n_P$) is plotted in orange.

SC and BE containers, which are then mapped to processing resources of the IPF system. Mixed-criticality in the ECU comes from auxiliary functions, such as logging and performance monitoring of the main perception function. Those auxiliary functions and the tasks implementing them are important for the perceived quality of the vehicle but not in terms of safety, and are usually of low criticality or not safety critical at all (ASIL A and QM). Perception, the main function, is considered to be classified as ASIL D given its importance for the vehicle. The ECU inherits then the highest ASIL from the functions it implements. In this use case, the IPF system must not only provide unprecedented computing power and efficiency, but it must also provide a flexible system architecture that supports frequent software updates and autonomously adapts to varying workloads, hazards and imminent hazards while meeting stringent requirements of critical ASILs (cf. Figure 7(b)).

VI. QUANTIFYING IPF'S RELIABILITY GAIN

We now investigate the reliability improvements achieved by applying the IPF paradigm with respect to permanent random hardware faults due to aging. We quantify the reliability by means of the well-known mean time to failure (MTTF) metric, which denotes the expected time to failure for a system [28]. The FIT rate, another relevant metric, can be derived from the MTTF with $\text{FIT} = 10^9 \cdot \text{MTTF}^{-1}$ [28].

We built a reliability model considering a hardware platform, as described in Section IV-A, where processing resources are tiles and the shared resource is the NoC. The model takes as input the number of processing ($r_P \in [1, 32]$) and shared resources ($r_S \in [1, 1]$), their respective failure rates ($\lambda_P \in [10^{-11}, 10^{-7}]$ and $\lambda_S \in [10^{-11}, 10^{-7}]$), and the number of critical containers ($n_P \in [1, 32]$). The critical containers are the containers whose failure cause the failure of the system. Considering that an IPF system fails if a requirement of the safety-critical workload is violated or, optionally, if the minimum QoS of the best-effort workload is violated, the critical containers consist of all SC containers and, optionally, the minimum set of BE containers required to meet the minimum QoS. Equation (1) shows the resulting MTTF equation of the reliability model.

$$MTTF = \sum_{x=n_P}^{r_P} \binom{r_P}{x} \sum_{l=0}^{r_P - x} \binom{r_P - x}{l} \frac{(-1)^l}{\lambda_S + x\lambda_P + l\lambda_P}. \quad (1)$$

We start by investigating the reliability gains obtained by increasing the degree of redundancy in an IPF system. Then we investigate further the trade-off between the robustness of the processing and shared resources, the robustness of individual processing resources impact the overall system robustness, and the impact of shared resources on the resulting reliability. Unless stated otherwise, the experiments consider $r_P = 4$, $r_S = 1$, $\lambda_P = 10^{-8}$, $\lambda_S = 10^{-8}$, and $n_P = 2$.

Figure 8 reports the MTTF of an IPF system when varying the degree of redundancy – i.e., varying the number of critical containers and the number of processing resources in a platform. With the proactive planning at runtime and the self-organization, IPF is able to handle multiple permanent error occurrences when redundant resources are available. In comparison with a baseline[4] system, IPF's MTTF increases between 4.77 and 1.8 times when doubling the number of processing resources from $r_P = 8$ to $r_P = 16$ (with $n_P = 8$) and $r_P = 2$ to $r_P = 4$ (with $n_P = 2$), respectively. By introducing a single spare processing resource into a baseline system, the reliability increases between 1.5 and 1.88 times for $r_P = n_P = 2$ and $r_P = n_P = 15$, respectively.

To put the results in perspective, let us consider the use cases from Section V. The pacemaker setup in the example of Figure 7 contains two critical containers and four processing resources ($n_P = 2, r_P = 4$). An IPF system, as in the example, can achieve an MTTF of $6 \cdot 10^7$ hours, an improvement by a factor of 1.8 over a baseline[4] (cf. Figure 8). Considering a single critical container accommodates the SC workload ($n_P = 1$), an IPF system can achieve MTTFs of $6.66 \cdot 10^7$ ($r_P = 2$) and $7.5 \cdot 10^7$ hours ($r_P = 3$), which is an increase of 33 and 50 percent in comparison with a baseline[4] system, respectively. That is beyond the 5-year life expectancy of the pacemaker, which is usually limited by the battery. The perception tasks of the automotive use case Section V-B would require a minimum of twenty containers to execute in time on an IPF system with thirty two processing resource clusters ($n_P = 20, r_P = 32$). An IPF system can achieve an MTTF of $3.9 \cdot 10^7$ hours (25.6 FIT), an improvement by a factor of 8.27 over a baseline[4] (cf. Figure 8). Although IPF is able to substantially extend the system lifetime, the use case still falls short of the 10^8 hours target for ASIL D. Next, we investigate why the MTTF struggles to surpass 10^8 hours.

Figure 9 reports the MTTF when varying the number of processing resources (r_P) and the failure rates of those resources (λ_P). Similar reliability levels can be achieved with different combinations of number of spare resources with different robustness levels: more less-reliable resources or fewer more-reliable resources. For example, an MTTF of $5 \cdot 10^7$ hours (20 FIT) can be achieved with two highly reliable

[4]Non-redundant or statically configured baseline system ($r_P = n_P$).

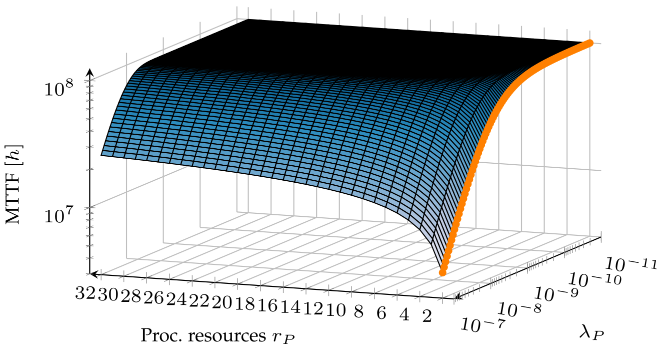

FIGURE 9. The redundancy-robustness trade-off of an IPF system in terms of MTTF. For reference, baseline ($r_P = 2$) is plotted in orange.

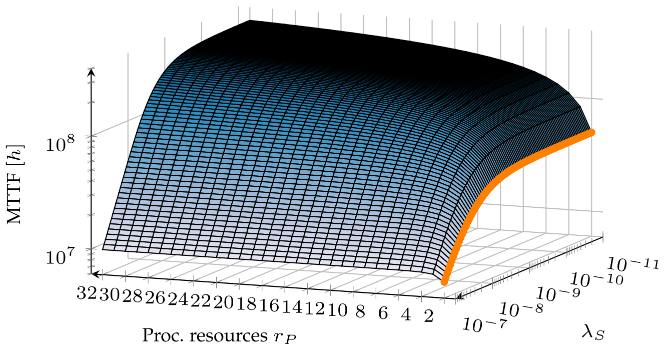

FIGURE 10. Impact of the shared resource on the reliability of an IPF system. For reference, baseline ($r_P = 2$) is plotted in orange.

cores, four medium-reliable cores, or even sixteen unreliable cores. However, note that increasing the degree of redundancy (r_P) indefinitely for a fixed λ_P brings limited benefit – e.g., $r_P > 8$ for $\lambda_P = 10^{-7}$. We found that to be caused by the use of available/spare resources in the IPF system to execute BE workload. When IPF allows BE workload to execute on any resource that is not required by the SC workload, that resource also ages, leading to the effect seen in Figure 9. When a permanent fault or imminent hazard occurs, a reconfiguration is triggered at the SyC, which performs a so-called *minimum repair*, after which all resources are "as good as old" since they are aging at approx. the same rate [28]. Thus, even though an IPF system can plan accordingly and handle multiple permanent error occurrences, which is possible as long as there are enough spare resources and valid ORs, the benefit can be limited depending on the resource usage policy.

Another observation is that, even when more robust resources are employed (smaller λ_P), a plateau is reached at approximately 10^8 h in Figure 9. That is caused by the shared resource, which can easily become the reliability bottleneck, as is the case of the interconnect [25], and limit the benefits of IPF's planning and self-organization. We then evaluated whether employing more-reliable shared resources would solve the problem. Figure 10 shows the MTTF when varying the number of processing resources (r_P) and the failure rate of the shared resource (λ_S). In contrast to Figure 9, we now fix $\lambda_P = 10^{-8}$ and vary λ_S. The results confirm the existence of the bottleneck – i.e., increasing the reliability of the shared resource (smaller λ_S) does increase the overall reliability. However, the reliability gains are still limited by the above-mentioned minimum repair effect.

In summary, the results show that the IPF paradigm is able to extend the system lifetime substantially, especially in larger systems such as the one in the automotive use case. However, the resource use policy requires special attention. Allowing all non-SC processing resources to be used for the execution of BE workload provides for higher BE performance but limits the system lifetime due to the more uniform resource aging (minimum repair effect). Thus, an envisioned, promising policy is to reserve a set of spare processing resources to be used only in case of imminent hazards. The size of that set can be derived from the target MTTF. Exploring this idea further is left for future work.

VII. RELATED WORK

This section summarizes the relevant state of the art on self-aware and mixed-critical embedded systems. We start with the definition of self-awareness of [13], whose survey gives a comprehensive overview of existing research on the topic. *Self-awareness* is defined as the combination of three properties that a system or service should possess: self-reflective, self-predictive, and self-adaptive. *Self-reflective* refers to being aware of the system state, its goals, and dynamic changes in them. *Self-predictive* refers to being able to predict dynamic changes (e.g., workload or requirements) and predict the effect of adaption actions; *Self-adaptive* refers to being able to proactively adapt and optimize to dynamic changes in order to meet QoS goals and requirements. Self-aware computing combines concepts that have been the subject of prior computer science research in areas including artificial intelligence, autonomic computing, self-adaptive and self-organizing systems, and cognitive computing [34]. In our work, we utilize aspects of computational self-awareness in our information processing factory to address four technical challenges of self-aware systems: learning, formulation of goals, scalability, ensuring correctness, and an appropriate design methodology.

Mixed-critical real-time systems are systems that implement functions of two or more distinct criticality levels, such as safety-critical and best-effort [10], [35]. Nowadays, most of the complex embedded systems found, e.g., in the automotive and avionics domains are evolving into mixed-critical systems to increase efficiency and reduce size, weight, and power (SWaP). The major challenge is enabling formal guarantees, such as performance and integrity, for a function or criticality independent of the other criticalities when they share resources. These systems have strict requirements [3], [11], [12] that call for dedicated techniques that assure safety without jeopardizing the efficiency of resource usage, the so-called partitioning-sharing trade-off [35]. A comprehensive overview of mixed-criticality is found in [35]. Traditionally, mixed-critical real-time systems

TABLE 2. IPF Comparison With State-of-the-Art Approaches

	[36]	[37]	[40]	IPF	DMR/TMR [28]
A. Self-reflective	∗	∗	✓	✓	
B. Self-predictive			✓	✓	
C. Self-adaptive	✓	✓	✓	✓	
1. Integrity	†	†		✓	✓
2. Reliability ↑	†	†		✓	†
3. Availability ↑	†			✓	∗
4. System-wide and hierarchical				✓	
5. Mixed-criticality aware				✓	✓
6. Variability aware			✓	✓	
7. Multi-objective		✓		✓	
8.Reconfiguration cost aware	✓	✓		✓	
9. Independent of design-time DSE			✓	✓	✓

Legend: ✓presents/improves, ∗ limited, † potentially.

have had static configurations for higher criticalities and flexibility for best-effort functions. There are also systems with room for adaptation [36], [37], which can be classified as self-adaptive systems. Self-awareness in mixed-critical real-time systems has received relatively little attention with approaches that focus on a specific aspect, such as communication [38]. The IPF paradigm for mixed-critical multiprocessing aims at a system-wide self-awareness that can autonomously handle and learn with changes in the environment, the system itself, and the implemented functions represented by the mixed-critical workload.

Employing system reconfiguration with task migration to overcome permanent errors and adapt to changes in the system is a recurrent idea. A comprehensive overview of trends in mapping on multi and many-core systems is found in [39]. A challenge that arises, however, when migrating mixed-critical workload since timing guarantees must be given to the safety-critical component of the workload not only before and after, but also during the migration process. Mapping reconfiguration of hard real-time tasks was recently investigated in [36] for NoC-based many-core systems. Similarly to IPF, the authors have considered the cost that reconfiguration incurs at runtime, which can violate the timing requirements of the safety-critical workload if not appropriately considered. The work in [36] relies on design-time design space exploration (DSE) to create a set of possible configurations for the system and a hybrid latency analysis for the migration, which combines a computation-intensive discovery of migration routes at design-time with a final latency calculation at runtime. Similarly, the authors in [37] propose a hybrid methodology, where design-time DSE is combined with an agent-based runtime adaptation, that targets dynamic cross-layer reliability. Although not explicitly addressed, [36], [37] can potentially be combined with mechanisms to ensure integrity, increase reliability and availability.

A summary of the relevant related work is given in Table 2 – intended as a feature comparison and not as an exhaustive list of features. In contrast to [36] and [37], IPF aims at becoming independent of design-time DSE in order to autonomously adapt to variations in the workload *and* in the physical system. Therefore, the proposed IPF paradigm introduces a planning entity that strategically performs DSE on demand and for a limited number of scenarios (the NORs) at a time, as opposed to requiring an exhaustive exploration at design-time. Moreover, IPF's self-reflective and self-predictive properties makes it variability aware. No two physical system instances behave or age exactly the same, even when executing the same workload, and that can only be addressed at runtime by observing the platform's operation. A similar trend is seen in other approaches, such as [40], which manages resources with self-awareness to achieve QoS targets in the presence of varying workload. For reference only, [40] is used here to represent a class of self-aware resource management approaches that address specific aspects other than reliability and are mixed-criticality unaware. In contrast to those, IPF adopts a system-wide, hierarchical, and multi-objective approach that can handle the different requirements and QoS targets of the mixed-critical workload. Finally, IPF is compared to the classic modular redundancy approaches (DMR and TMR) [28]. Both are widely used in industrial safety-critical systems to provide error detection and achieve integrity, e.g., with lock-step execution [26]. However, they can only detect immediate errors – that is, once the error affects the system's state – as opposed to IPF's proposed imminent hazard detection and handling. Despite its expressive cost, modular redundancy does not extend the average system lifetime (reliability) in its classical form [28]. It can, however, increase reliability in conjunction with other approaches, as it is done in IPF. IPF's self-aware resource management exploits the inherent redundancy of resources in multi and many-core systems to effectively extend the system's lifetime.

The IPF paradigm combines different techniques into a self-aware factory whose autonomy accounts for the various constraints that mixed-criticality imposes on the system. Some of those concepts have been discussed in a special session [9] that occurred between an early version of this paper [41] and its publication. In summary, this paper extends and supersedes [41], and it elaborates on and integrates the mechanisms discussed in [9], reusing the illustrative example of Figure 7.

VIII. CONCLUSION

This paper introduced the self-aware IPF paradigm for mixed-critical embedded systems. IPF's self-aware, dynamic management of the system enables the system to plan and handle changes to the environment, the workload, and to the system itself at runtime as they occur. A five-layer hierarchical organization was introduced with a system configuration framework based on operating regions and operating points that enables self-awareness, self-diagnosis, self-organization and self-optimization in mixed-criticality. An invariant-based safety argument that enables the use of IPF in mixed-criticality was developed, and provides the underlying assumptions that must be fulfilled when applying the paradigm. The application of IPF towards long-term dependability was illustrated with two representative use cases from heterogeneous domains. The new concept of imminent hazards was introduced, which allows risks in the system to be proactively handled. Finally, an experimental evaluation makes an initial assessment of the achievable reliability gains when using IPF for long-term dependability, and highlights bottlenecks and opportunities to be explored.

ACKNOWLEDGMENTS

We acknowledge financial support from the following: NSF Grant CCF-1704859; DFG Grants ER168/32-1 and HE4584/7-1.

REFERENCES

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th ed. San Francisco, CA, USA: Morgan Kaufmann, 2017.

[2] R. Gaillard, *Single Event Effects: Mechanisms and Classification*. Boston, MA, USA: Springer, 2011, pp. 27–54.

[3] "ISO 26262: Road vehicles – functional safety," International Standards Organization, 2018.

[4] A. Herkersdorf *et al.*, "Resilience articulation point (RAP): Cross-layer dependability modeling for nanometer system-on-chip resilience," *Microelectronics Rel.*, vol. 54, no. 6–7, pp. 1066–1074, 2014.

[5] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annu. Int. Symp. Comput. Architecture*, 2011, pp. 365–376.

[6] J. Henkel, H. Khdr, S. Pagani, and M. Shafique, "New trends in dark silicon," in *Proc. 52nd ACM/EDAC/IEEE Des. Autom. Conf.*, 2015, pp. 1–6.

[7] N. Dutt, F. J. Kurdahi, R. Ernst, and A. Herkersdorf, "Conquering MPSoC complexity with principles of a self-aware information processing factory," in *Proc. 11th IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codesign System Synthesis*, 2016, pp. 1–4.

[8] A. Sadighi *et al.*, "Design methodologies for enabling self-awareness in autonomous systems," in *Proc. Des. Autom. Test Europe Conf.*, 2018, pp. 1532–1537.

[9] E. A. Rambo *et al.*, "The information processing factory: A paradigm for life cycle management of dependable systems," in *Proc. 14th IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codesign Syst. Synthesis*, 2019, pp. 1–10.

[10] R. Ernst and M. Di Natale, "Mixed criticality systems–a history of misconceptions?" *IEEE Design Test*, vol. 33, no. 5, pp. 65–74, Oct. 2016.

[11] "IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, ed.2.0," International Electrotechnical Commission, 2010.

[12] "DO-254: Design assurance guidance for airborne electronic hardware," RTCA Incorporated, 2000.

[13] A. Jantsch, N. Dutt, and A. M. Rahmani, "Self-awareness in systems on chip– a survey," *IEEE Design Test*, vol. 34, no. 6, pp. 8–26, Dec. 2017.

[14] "ANSI/ISA-95.00.01–2010 (IEC 62264-1 mod) - enterprise-control system integration - part 1: Models and terminology," International Society of Automation, 2010.

[15] M. Seo and F. Kudarhi, "Efficient tracing methodology using automata processor," in *Proc. 14th IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codesign Syst. Synthesis*, 2019, Art. no. 80.

[16] J. Zeppenfeld, A. Bouajila, W. Stechele, and A. Herkersdorf, "Learning classifier tables for autonomic systems on chip," *GI Jahrestagung*, vol. 134, pp. 771–778, 2008.

[17] S. W. Wilson, "ZCS: A zeroth level classifier system," *Evol. Comput.*, vol. 2, no. 1, pp. 1–18, 1994.

[18] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis–the symta/s approach," *IEE Proc.-Comput. Digital Tech.*, vol. 152, no. 2, pp. 148–166, Mar. 2005.

[19] R. Alur, "Timed automata," in *Proc. Int. Conf. Comput. Aided Verification*, 1999, pp. 8–22.

[20] B. Motruk, J. Diemer, R. Buchty, R. Ernst, and M. Berekovic, "Idamc: A many-core platform with run-time monitoring for mixed-criticality," in *Proc. IEEE 14th Int. Symp. High-Assurance Syst. Eng.*, 2012, pp. 24–31.

[21] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, First Quarter 2004.

[22] C. Constantinescu, "Trends and challenges in vlsi circuit reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, Jul./Aug. 2003.

[23] E. Landman *et al.*, "Degradation monitoring – from a vision to reality," in *Proc. IEEE Int. Rel. Phys. Symp.*, 2019, pp. 1–4.

[24] E. A. Rambo and R. Ernst, "Replica-aware co-scheduling for mixed-criticality," in *Proc. 29th Euromicro Conf. Real-Time Syst.*, 2017, pp. 20:1–20:20.

[25] E. A. Rambo, Y. Shang, and R. Ernst, "Providing integrity in real-time networks-on-chip," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1907–1920, Aug. 2019.

[26] Infineon, "32-bit TriCore^{TM} AURIX^{TM} – tc3xx," 2019. [Online]. Available: https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc3xx/

[27] B. Döbel, H. Härtig, and M. Engel, "Operating system support for redundant multithreading," in *Proc. 10th ACM Int. Conf. Embedded Softw.*, 2012, pp. 83–92.

[28] A. Hùyland and M. Rausand, *System Reliability Theory: Models and Statistical Methods*, vol. 420, Hoboken, NJ, USA: Wiley, 2009.

[29] "Classic Platform – release 4.4.0," AUTOSAR, 2018.

[30] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, "Development of autonomous carpart II: A case study on the implementation of an autonomous driving system based on distributed architecture," *IEEE Trans. Ind. Electron.*, vol. 62, no. 8, pp. 5119–5132, Aug. 2015.

[31] T. Steinbach, "Ethernet-based network architectures for future real-time systems in the car," *ATZ Worldwide*, vol. 121, no. 7–8, pp. 72–77, 2019.

[32] P. Wang, X. Huang, X. Cheng, D. Zhou, Q. Geng, and R. Yang, "The apolloscape open dataset for autonomous driving and its application," in *IEEE Trans. Pattern Anal. Mach. Intell.*, to be published, [Online]. Available: https://doi.org/10.1109/TPAMI.2019.2926463, doi: 10.1109/TPAMI.2019.2926463.

[33] W. Lim, S. Lee, M. Sunwoo, and K. Jo, "Hierarchical trajectory planning of an autonomous car based on the integration of a sampling and an optimization method," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 2, pp. 613–626, Feb. 2018.

[34] S. Kounev *et al.*, *The Notion of Self-aware Computing*. Cham, Switzerland: Springer, 2017.

[35] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surv.*, vol. 50, no. 6, 2018, Art. no. 82.

[36] B. Pourmohseni, S. Wildermann, M. Glaû, and J. Teich, "Hard real-time application mapping reconfiguration for NoC-based many-core systems," *Real-Time Syst.*, vol. 55, pp. 1–37, 2019.

[37] S. S. Sahoo, B. Veeravalli, and A. Kumar, "A hybrid agent-based design methodology for dynamic cross-layer reliability in heterogeneous embedded systems," in *Proc. 55th ACM/EDAC/IEEE Des. Autom. Conf.*, 2019, pp. 1–6.

[38] A. Kostrzewa, S. Tobuschat, and R. Ernst, "Self-aware network-on-chip control in real-time systems," *IEEE Design Test*, vol. 35, no. 5, pp. 19–27, Oct. 2018.

[39] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc. 50th ACM/EDAC/IEEE Design Autom. Conf.*, 2013, pp. 1–10.

[40] Y. Song, O. Alavoine, and B. Lin, "A self-aware resource management framework for heterogeneous multicore SoCs with diverse QoS targets," *ACM Trans. Architecture Code Optim.*, vol. 16, no. 2, pp. 1–23, 2019.

[41] E. A. Rambo *et al.*, "The information processing factory: Organization, terminology, and definitions," 2019. [Online]. Available: https://arxiv.org/abs/1907.01578

EBERLE A. RAMBO (Member, IEEE) received the BS and MS degrees in computer science from the Federal University of Santa Catarina, Florianópolis, Brazil, in 2009 and 2011, respectively, and the Dr.-Ing. degree in electrical engineering from the Braunschweig University of Technology (TUBS), Braunschweig, Germany, in 2019. He is currently a postdoctoral researcher with TUBS. His research interests include embedded systems, mixed-criticality, and self-awareness.

BRYAN DONYANAVARD received the PhD degree in computer science from the University of California, Irvine, in 2019, and he is currently a researcher with the Center for Embedded and Cyber-physical Systems (CECS), University of California, Irvine. His research interest inclues generally in intelligent systems management.

MINJUN SEO received the BE and MS degrees from Kyungnam University, and the PhD degree from the University of Arizona, in 2018. He is currently a postdoctoral researcher with the Center for Embedded Cyber-physical Systems (CECS). His research interests include runtime verification, hardware-based monitoring, and non-intrusive tracing of systems.

FLORIAN MAURER received the BSc and MSc degrees from the Technical University of Munich, Munich, Germany, in 2016 and 2018, respectively. He is currently working toward the PhD degree in electrical engineering at the Technical University of Munich, Munich, Germany. His research interests include self-aware and self-adaptive multi- and many-core systems.

THAWRA KADEED received the BS and MSc degrees in computer engineering from the Aleppo University, Syria, in 2009 and 2012, respectively. She is currently working toward the PhD degree at the Braunschweig University of Technology, Braunschweig, Germany. Her research interests include embedded system design for mixed-critical real-time applications, focusing on self-aware systems through Networks-on-chip (NoCs) level energy management and system level errors handling.

CAIO B. DE MELO received the BS and MS degrees from the University of Brasǒlia, in Brazil. He is currently working toward the PhD degree in computer science with the University of California, Irvine.

BISWADIP MAITY received the BS degree from Jadavpur University. He is currently working toward the PhD degree in computer science at the University of California, Irvine His research interests include embedded systems, internet of things, and multi-core architectures.

ANMOL SURHONNE received the BE degree from the PES Institute of Technology, and the MSc degree from Nanyang Technological University and Technical University of Munich. He is currently working toward the PhD degree in electrical and computer engineering at the Technical University of Munich. His research interests include self aware multi/many core systems and machine learning.

ANDREAS HERKERSDORF received the Dr. degree from ETH Zurich, Switzerland, in 1991. He is a professor with the Department of Electrical and Computer Engineering, Technical University of Munich (TUM). Between 1988 and 2003, he has been with the IBM Research Laboratory, Rueschlikon, Switzerland. Since 2003, he leads the chair of Integrated Systems at TUM. His research interests include application-specific multicore-processor architectures, IP network processing, network-on-chip, and self-adaptive, fault-tolerant computing.

FADI KURDAHI (Fellow, IEEE) received the PhD degree from the University of Southern California, in 1987. Since then, he has been a faculty with the Department of Electrical Engineering and Computer Science, University of California, Irvine, where he conducts research interests include Computer-Aided Design and design methodology of large scale systems. He currently serves as the associate dean for graduate and professional Sstudies of the Samueli School of Engineering, and the director of the Center for Embedded & Cyber-physical Systems (CECS), comprised of world-class researchers in the general area of Embedded and Cyber-physical Systems. He is the American Association for the Advancement of Science (AAAS).

NIKIL DUTT (Fellow, IEEE) received the PhD degree in computer science from the University of Illinois at Urbana-Champaign, in 1989, and is currently a distinguished professor of Computer Science, Cognitive Sciences, and EECS with the University of California, Irvine. His research interests include embedded systems, electronic design automation (EDA), computer systems architecture and software, healthcare IoT, and brain-inspired architectures and computing. He is a fellow of the ACM and recipient of the IFIP Silver Core Award.

ROLF ERNST (Fellow, IEEE) received the diploma degree in computer science and the Dr.-Ing. degree in electrical engineering from the University of Erlangen-Nuremberg, Erlangen, Germany, in 1981 and 1987, respectively. From 1988 to 1989, he was with Bell Laboratories, Allentown, PA. Since 1990, he has been a professor of Electrical Engineering at Braunschweig University of Technology, Braunschweig, Germany. His research activities include embedded system design and design automation.